vectorizing across entire training set.

iterative approach:

for i = 1 to m.

$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$

$W^{[1]}$ shape = (k, n_x)
k sets of weights. (w), where $w \in \mathbb{R}^{n_x}$, k equals number of neurals at this hidden layer.

$a^{[1](i)} = \sigma(z^{[1](i)})$

$z^{[2](i)} = W^{[2]} x^{(i)} + b^{[2]}$

$a^{[2](i)} = \sigma(z^{[2](i)})$

Vectorized Algorithm:

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]} X + b^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

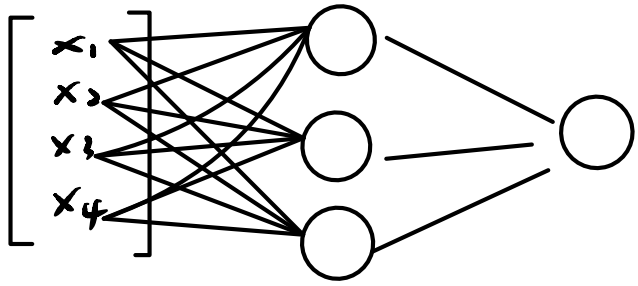
$z^{[1](1)} = W^{[1]} x^{(1)} + \cancel{b^{[1]}}^{0}$ $z^{[1](2)} = W^{[1]} x^{(2)} + \cancel{b^{[1]}}^{0}$ $\cdots$ $z^{[1](m)} = W^{[1]} x^{(3)} + \cancel{b^{[1]}}^{0}$

$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix}$ $x^{(1)}: \begin{bmatrix} | \\ | \end{bmatrix} \Rightarrow W^{[1]} x^{(1)} = \begin{bmatrix} | \\ | \end{bmatrix}$ $W^{[1]} x^{(2)} = \begin{bmatrix} | \\ | \end{bmatrix} \cdots$

$$\Rightarrow W^{[1]} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix} = Z^{[1]}$$

My intuition:

Considering a training set containing only 1 datapoint $\in \mathbb{R}^4$, and a neural network with 1 hidden layer, containing three neutrons.

→ X: 

output.

Then: what does the weights W in hidden layer actually do on the input data (data coming from prev layer)

W: $\begin{bmatrix} — & w^{(1)} & — \\ — & w^{(2)} & — \\ — & w^{(3)} & — \end{bmatrix}$ $\begin{bmatrix} | \\ x^{(1)} \\ | \end{bmatrix}$ $\Rightarrow$ $\begin{bmatrix} | \\ wx^{(1)} \\ | \end{bmatrix}$

3×4 $\quad$ 4×1 $\quad$ 3×1

W is actually a transformation matrix that transforms $x^{(1)}$ from a 4-D vector to a 3-D vector, which equals the number of neurons in hidden layer. (one output per hidden neuron).

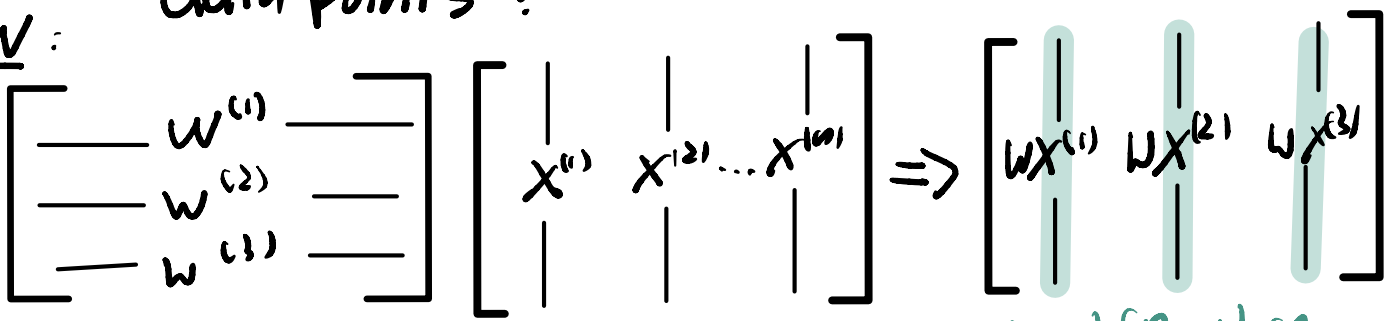
if instead of having 3 neurons, we have 10, Then weights W will be in shape: 10×3. and $Wx^{(1)}$ will transform $x^{(1)}$ from 4-D vector into a 10-D vector.

This intuition is mainly caused by the fact that each neuron in neural network, generally speaking, always only output one output signal.

Due to this nature of the neural network, mathmatically speaking, each layer's weights $\underline{W}$ is essentially a transformation matrix that transform data coming from prev layer into a n-Dimension vector, where n equals the amount of neurons at the current layer.

Then: what would happen if X contains more than one data points ?

$\underline{W}$:



$(\underline{3}, n_x)$ (n_x, m)

$(n_w, 1)$ $(n_w, 1)$ $(n_w, 1)$

$\underline{n_w}$ = # of neurons at this layer.

=> $Z : (\underline{3}, n_x)$

- Similar transformation effect on each $\underline{x^{(i)}}$.